Option informatique : Corrigé du Devoir Surveillé n° 1

Exercice 1

(a)

- `[| true, 1.0; false, 2e8 |]`
- `fun v i -> snd v.(i)`
- `fun f g x -> g (f x)`

(b)

- `int`
- `(int * string) list list`
- `(int * bool * float) option list`

(c)

- `13`
- `[false; true; false]`
- `[7; 10]`

Exercice 2

(a)

```
let rec pgcd a b = match b with
  | 0 -> a
  | _ when b>a -> pgcd b a
  | _ -> pgcd b (a mod b) ;;
```

Cette fonction termine car son second argument décroît strictement à chaque appel, et reste supérieur à 0.

(b)

```
let pgcd2 a0 b0 =
  let a = ref (max a0 b0)
  and b = ref (min a0 b0) in
  while !b > 0 do
    let r = !a mod !b in
    a := !b;
    b := r
  done;
  !a ;;
```

La boucle termine pour la même raison : `!b` décroît strictement à chaque itération.

Exercice 3

```
let rec nombreDeUns n = match n with
  | 0 -> 0
  | _ when n mod 2 = 0 -> nombreDeUns (n/2)
  | _ -> 1 + nombreDeUns (n/2);;

let rec binaire n = match n with
  | 0 -> []
  | 1 -> [true]
  | _ when n mod 2 = 0 -> false :: binaire (n/2)
  | _ -> true :: binaire (n/2);;

let palindrome i = binaire i = rev (binaire i);;

let rec decimal l = match l with
  | [] -> 0
  | true::l' -> 1 + 2 * decimal l'
  | false::l' -> 2 * decimal l';;

for i = 0 to 100 do
  if i <> decimal (binaire i)
  then failwith ("erreur pour " ^ string_of_int i)
done;;

let rec ou_exclusif l1 l2 = match l1, l2 with
  | [], _ -> l2
  | _, [] -> l1
  | x :: l1', y :: l2' ->
    match x, y with
    | true, true
    | false, false -> false :: ou_exclusif l1' l2'
    | true, false
    | false, true -> true :: ou_exclusif l1' l2'
;;

let ou_exclusif_int i j =
  decimal (ou_exclusif (binaire i) (binaire j));;
```

Exercice 4

```
let rec merge l1 l2 = match l1, l2 with
  | [], _ -> l2
  | _, [] -> l1
  | x::l1', y::l2' ->
    if x < y then x :: merge l1' l2
```

```
    else y :: merge l1 l2'   (* x >= y *)
;;

merge [1;2;3;5;8] [2;3;7;8;9] = [1;2;2;3;3;5;7;8;8;9];;
merge [] [1;2;3] = [1;2;3];;
merge [1;2;3] [3;4] = [1;2;3;3;4];;

let rec entrelacer sep l = match l with
  | [] -> []
  | [x] -> [x]
  | x::y::l' -> x :: sep :: entrelacer sep (y :: l');;

entrelacer "," ["a"; "b"; "c"] = ["a"; ","; "b"; ","; "c"];;
entrelacer 12 [] = [];;

let mapi f l =
  let rec mapi_aux f i l = match l with
    | [] -> []
    | x :: l' -> (f i x) :: mapi_aux f (i+1) l'
  in
  mapi_aux f 0 l;;

(* fonction auxiliaire pour subset *)
let rec mem x l = match l with
  | [] -> false
  | y :: l' -> x = y || mem x l' ;;

let rec subset l1 l2 = match l1 with
  | [] -> true
  | x :: l1' -> mem x l2 && subset l1' l2;;

subset [1;2] [3;1;4;2;5] = true;;
subset [] [42] = true;;
subset [1;2;3;4] [2;4;3;0;10] = false;;

let rec subset_trie l1 l2 = match l1, l2 with
  | [], _ -> true
  | _, [] -> false
  | x::l1', y::l2' ->
    if x < y then false   (* x ne peut apparaitre dans l2 *)
    else if x = y then subset_trie l1' l2'
    else subset_trie l1 l2'  (* sauter y *)
;;
```

Exercice 5

```
let MAXVAL = 30;;

let compter a count =
  for i = 0 to vect_length a - 1 do
    let x = a.(i) in
    count.(x) <- count.(x) + 1;
  done
;;

let trier a =
  let count = make_vect (MAXVAL+1) 0 in
  compter a count;
  (* reconstruire le tableau trie *)
  let result = make_vect (vect_length a) 0 in
  let k = ref 0 in  (* indice courant dans result *)
  for i = 0 to MAXVAL do
    (* ajouter i dans result, count.(i) fois *)
    for j = 1 to count.(i) do
      result.(!k) <- i;
      incr k
    done;
  done;
  result
;;

trier [| 1; 2; 0; 10; 2 |] = [| 0; 1; 2; 2; 10 |];;
trier [| 16; 12; 3; 0; 11 |] = [| 0; 3; 11; 12; 16 |];;
```